Первое практическое домашнее задание.
Дедлайн: 19.02.2023 23:59:59

Требования к графу.

Так как граф это математическая абстракция, его можно параметризовать как угодно. Более того, это множество вершин и пар вершин aka рёбер.

1. Сделать абстрактный класс граф. Параметризовать его типом вершины (по умолчанию это ее номер) и типом ребра (по умолчанию это пара вершин).
2. От графа отнаследовать две реализации: на списке смежности и на матрице смежности.
3. Граф должен уметь.
   - Отдавать число вершин и ребер.
   - Отдавать по вершине список соседей (в вершинах).
   - Отдавать по вершине итератор на список соседей. Должен работать `range-based for`.
   - (По желанию) Сделать аналог `filter iterator`, который будет способен фильтровать ребра в зависимости от условия.
4. Прописать юзинги `vertex_type`, `edge_type`.

Определение. Коллбэком (callback) в данном контексте будем называть некий функтор (объект, у которого есть оператор вызова `()`), который каждое необходимое нам событие будет как-то обрабатывать.

Пример. В задаче 1G можно его реализовать так, что он хранит в себе (но не копирует, например, хранит ссылку или указатель) отображение из ребра в его номер. Тогда на каждое нахождение моста можно будет вызывать данный коллбэк и отдавать его номер в входных данных.

Требования к задаче на 1A.

1. Необходимо реализовать паттерн Visitor (параметризуется типами вершин и ребер). Вот пример, на который будет опора: boost. Необходимо реализовать абстрактный визитор (вам понадобятся не все методы из документации). Надо будет реализовать его наследника, который сможет для каждой вершины сохранить вершину предка. Тогда восстановить путь достаточно просто.
2. Ваш алгоритм (удобно сделать классом) должен уметь принимать граф и визитор, а возвращать вектор вершин (шаблонных), что лежат на пути.
3. По идее подсчет веса пути должен уметь общаться с графом, однако на текущий момент опустим это, так как это его длина. Надо реализовать получение веса пути как-нибудь (конечно в идеале чтобы можно было вызвать `std::accumulate` от результата).
4. В идеале конечно, может потребоваться возвращать не вектор вершин, а что-то другое, для этого надо будет делать `Callbacks`. Однако сейчас об этом можно не думать.

Требования к задаче на 1G.

1. Необходимо реализовать паттерн Visitor (параметризуется типами вершин и ребер). Вот пример, на который будет опора: boost. Необходимо реализовать абстрактный визитор (вам понадобятся не все методы из документации). Надо будет реализовать его наследника, который сможет для каждого ребра (шаблонного) сказать, является ли он мостом.
2. Ваш алгоритм (удобно сделать классом) должен уметь принимать граф и визитор, а возвращать вектор ребер, что являются мостами.
3. В идеале конечно, может потребоваться возвращать не вектор номеров ребер, а что-то другое (например, множество тех ребер, чьи вершины от старта не далее k ребер от какой-то вершины), для этого надо будет делать `Callbacks`. Однако сейчас об этом можно не думать.